A Proof-of-Concept RISC-V with 128-bit Extension

Frédéric Pétrot

Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP*, TIMA, F-38000 Grenoble, France

frederic.petrot@univ-grenoble-alpes.fr

*Institute of Engineering Univ. Grenoble Alpes

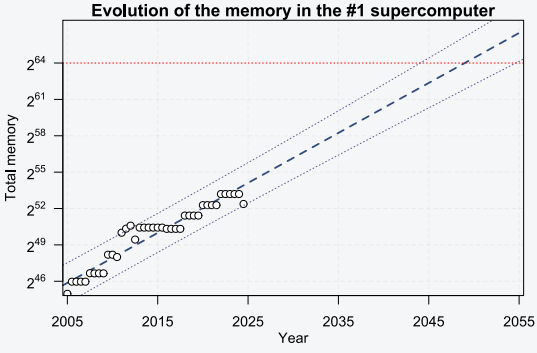
Why 128-bit?

No really necessary for data

- ≈80% of elementary integer data is ≤ 32-bit on 64-bit general purpose machines
- Quad-float useful in rare circumstances

But maybe for addresses: towards hundreds of exabytes

- HPC clusters
  + 1 bit every other year
  Not so far from exabyte (2^{60})
  (ORNL reached exaflop in 2022)
- We'll run out of addresses by 2030 !
  If the whole memory and disk is shared and byte addressable
  Otherwise, it's predicted to be 2049



128-bit RISC-V specification

Principle : `xlen` = $2^{\texttt{MXL}+4}$ $\rightsquigarrow$ `MXL`, field of `misa` equals 3
Registers, general purpose or control and status, extended to 128-bit

Impact on instruction set

- Usual instructions work on 128-bit
- Instructions with `w` suffix work on 32-bits
- Instructions with `d` suffix work on 64-bits
- `lq` and `sq` access 128-bit at once (No alignment constraints)
- Shifts by constants: 5, 6, or 7 bits in insn encoding

Additional instructions compared to rv64im

- 3 memory accesses `lq`, `sq`, `ldu`
- 3 adds: `addd`, `addid`, `subd`
- 6 shifts `slld`, `srld`, `srad`, `sllid`, `srlid`, `sraid`
- 5 mult/div/rem: `muld`, `divd`, `divud`, `remd`, `remud`

Cross-development environment

What does exist for 128-bit architectures support?

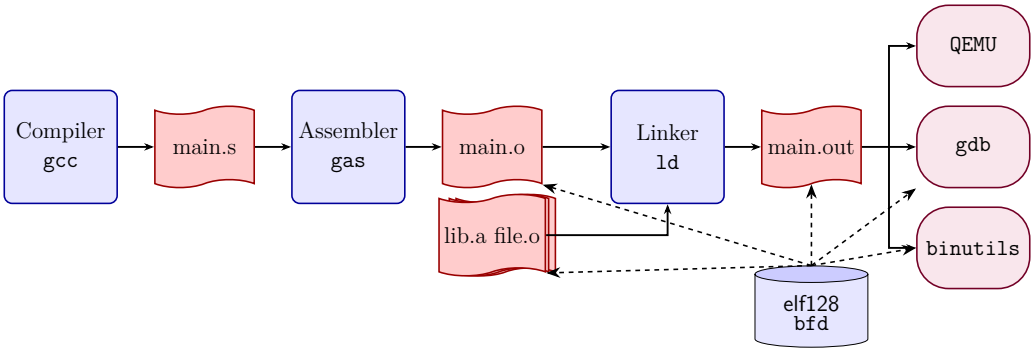
Nothing, or not far from it!

- No existing 128-bit object or executable format
- Many tools/libraries concerned
- Quite “sensible” code base

`gcc` vs `llvm`?

`gcc`, for legacy reasons, with GNU’s `binutils`

Cross-development environment



Elf128 ABI Proposal

C Language Data Model 128-bit extension

Data Type	ILP32	LP64	LLP128
`char`	8	8	8
`short`	16	16	16
`int`	32	32	32
`long`	32	64	64
`long long`	64	64	128
`void *`	32	64	128

Name	Size	Algnt	Purpose
`Elf128_Addr`	16	16	Unsigned program address
`Elf128_Off`	16	16	Unsigned file offset
`Elf128_Half`	2	2	Unsigned medium integer
`Elf128_Word`	4	4	Unsigned integer
`Elf128_Sword`	4	4	Signed integer
`Elf128_Xword`	8	8	Unsigned long integer
`Elf128_Sxword`	8	8	Signed long integer
`Elf128_Xxword`	16	16	Unsigned long long integer
`Elf128_Sxxword`	16	16	Signed long long integer
`unsigned char`	1	1	Unsigned small integer

Good News

- Extending the spec is pretty straightforward
  Just adding `Elf128_Xxword` and `Elf128_Sxxword`
- Quite a few paddings needed to warranty alignment, though

Elf128 ABI Support in binutils

Work done on GNU `binutils-gdb`

- `configure`: Complex tools configurations to support 128-bit targets
  A headache for rebuilding from scratch due to scripts pre-generation
- `bfd`: Support of the Elf128 file format
- `opcodes`: Addition of new insns and immediates (for constant shifts)
- `binutils`: Link to `libbfd.a` and `opcodes`, plus a few hacks
- `gas`: Addition of new insns and imms, already supports `.octa` for 16-bytes consts
- `ld`: Relocations/offsets on 128 bits
- `gdb`: 128-bit general purpose registers, computations, communication and display

Overall

- Intricate, but well architected, piece of software

C Data Model Support in gcc

Handling 128-bit in gcc

- 128-bit variables exist in gcc/llvm, but 128-bit constants do not!:

```
__int128_t x; /* Ok */
__uint128_t y; /* Ok */
__uint128_t x = 0xdeafbeefdeadbeefdeadbeefdeadbeef; /* Ko */
```

- 128-bit `long long` and `void *` supported through `__int128_t` and `__uint128_t`
- Code generation making use of the new insns:
  - for signed and unsigned `long` arithmetic and logic operations (`d` suffix)
  - for 128-bit memory accesses (`q` suffix)

Overall

- Intricate, but well architected, piece of software too!

QEMU 128-bit support on 64-bit hosts

Arith/Shift/Cmp: Lots of bit trickery!

Given (a_h, a_l) and (b_h, b_l) two 128-bit values build from 2 64-bit values

Add: $r_l \leftarrow a_l + b_l; r_h \leftarrow a_h + b_h + r_l \overset{u}{<} a_l$

$\Rightarrow$ 1 µ-op and only 2 x86-64 insns thanks to `adc`, soon 5 to 10 µ-ops for other insns though

Mult hi/lo, signed/unsigned

Arithmetic trick avoiding taking the absolute values of the operands*

$\Rightarrow$ 5 to 10 µ-ops depending on the exact instruction

Div/Rem

C function called from within Translation Blocks, a.k.a *Helpers*

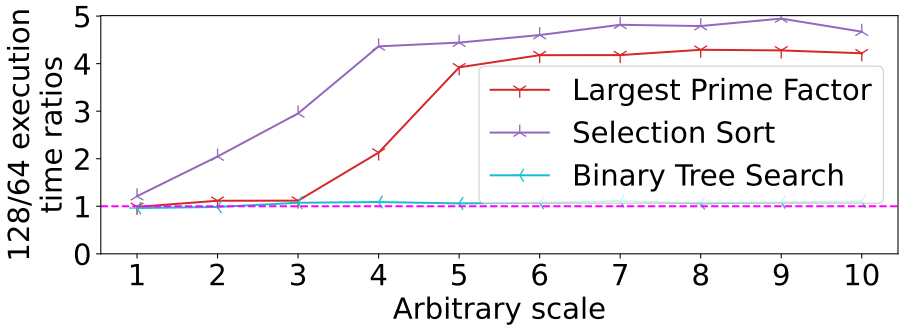
$\Rightarrow$ Call overhead ($\approx$ 10 x86-64 insns) amortized by C optimization vs µ-op optimizer

Load/Store

$2\times$ the number of accesses, with address increment and virt to phys translation

* See Hacker's Delight, Fig. 8.1, page 173.

QEMU performances on simple synthetic benchmarks



Execution time: 128-bit over 64-bit ratio

128-bit CVA6 Core

Should be as simple as `localparam CVA6ConfigXlen = 128;`

Usual patterns:

```
if (CVA6Cfg.XLEN == 64) begin
...
end else begin
...
end
V = CVA6Cfg.IS_XLEN64 ? 6 : 5;
P = (CVA6Cfg.XLEN == 64) ? 3 : 2;
if (CVA6Cfg.IS_XLEN32) begin
...
end
```

Not a criticism!
Useful and nice piece of code, but:

- Some refactoring needed upfront
- Lots of small pieces of code in many places
- Works only with HPDcache
- Verification environment not so simple
- Path to layout still needs work

Wrap-up

Status

- Available Cross-dev tools
  ⇒ With `newlib` but no `glibc`
  ⇒ Without much 128-bit specific optimisations in `gcc`
  ⇒ But with probably lots of bugs!
- rv128 in upstream QEMU since Jan 2022, with mttcg since Mar 2026
  `qemu-system-riscv64 -cpu x-rv128 ...`
- 128-bit CVA6 core running small bare metal benchmarks
  Part of privileged spec to be defined for 128-bit
  ⇒ Page-Table organization

Thanks for listening

Thanks to:

- The ANR and the partners of the ANR Maplurinum Project
- QEMU developers, in particular A. Francis, R. Henderson, and Ph. Mathieu-Daudé
- My undergrad students that contributed to that weird topic
- César Fuguet for his help with HPDcache and riscvbarelib

Dockerfile gathering the developments

`https://github.com/fpetrot/128-bit-riscv-devs`